On Finding Joint Subspace Boolean Matrix Factorizations

Pauli Miettinen*

Abstract

Finding latent factors of the data using matrix factorizations is a tried-and-tested approach in data mining. But finding shared factors over multiple matrices is more novel problem. Specifically, given two matrices, we want to find a set of factors shared by these two matrices and sets of factors specific for the matrices. Not only does such decomposition reveal what is common between the two matrices, it also eliminates the need of explaining that common part twice, thus concentrating the non-shared factors to uniquely specific parts of the data. This paper studies a problem called Joint Subspace Boolean Matrix Factorization asking exactly that: a set of shared factors and sets of specific factors. Furthermore, the matrix factorization is based on the Boolean arithmetic. This restricts the presented approach suitable to only binary matrices. The benefits, however, include much sparser factor matrices and greater interpretability of the results. The paper presents three algorithms for finding the Joint Subspace Boolean Matrix Factorization, an MDL-based method for selecting the subspaces' dimensionality, and throughout experimental evaluation of the proposed algorithms.

1 Introduction

Consider the following problem: We are given two sets of tagged images with tags from a common collection. We know that these tags are rather precise (for example, if an image contains a building, it typically has tags such as 'building', 'door', and 'window'). But the two sets do not contain same images, and are also tagged by different people. Our aim is two-fold: First, we want to reduce the number of tags by using latent super-tags (in the above example, we could replace the three tags with 'building' super-tag, as most buildings indeed have doors and windows). Second, we want to know which of these super-tags appear in both image sets, and which are specific to one of them.

A conventional answer would be to represent the image–tag information as two matrices, and decompose these matrices into factors in order to find the super-tags. But this approach has few problems. First, if we use conventional matrix factorization methods such as SVD or NMF, we have to define our super-tags differently. With SVD, we only project the original tags into some lower-dimensional space, but this lower-dimensional space on itself is not enough to define which tags 'go together'. With NMF the lower-dimensional space is restricted to the nonnegative orthant, which helps as the factor can be interpret as giving weights to tags' commitment to the super-tags, but does not solve the problem fully. For example, if tag t is in supertag s_1 by weight 0.8 and in super-tag s_2 by weight 0.7, an image that has both of these super-tags would have tag t by weight 1.5. And if you lower the weights, you also lower the weight of tag t in images with only one of these super-tags.

The other problem is that doing the decomposition does not tell which super-tags are shared and which are specific. We could decompose the matrices independently and call similar supertags shared, and dissimilar specific. But then we have to decide which similarity measure to use, and how similar is similar enough. Moreover, it is quite possible that the super-tags are not similar at all: it might be that the differences in the tagging (or images) make different types of super-tags optimal for the different image sets.

The purpose of this paper is to propose a solution that overcomes both of these problems: the super-tags are uniquely defined and the shared and specific supertags are found simultaneously. This is achieved by merging two earlier approaches, the Boolean Matrix Factorization and Joint Subspace Matrix Factorization, into Joint Subspace Boolean Matrix Factorization. The paper will present the existing approaches and the proposed approach (Section 3) and algorithms for finding the Joint Subspace Boolean Matrix Factorization (Section 4). The algorithms require three parameters, namely the dimensionalities of shared and specific subspaces, and setting these correctly is not a trivial task. Therefore, Section 5 presents a method for automatically evaluating the best parameter combination using the Minimum Description Length Principle. The proposed algorithms are tested extensively in Section 6.

2 Notation and Terminology

If $\boldsymbol{A}$ is an n-by-m binary matrix, $|\boldsymbol{A}|$ denotes the number of 1s in it, i.e. $|\boldsymbol{A}| = \sum_{i,j} a_{ij}$. The *sum of absolute differences* between two binary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is $|\boldsymbol{A} - \boldsymbol{B}|$. The squared Frobenius norm of an arbitrary matrix $\boldsymbol{A}$, $\|\boldsymbol{A}\|_F^2$, is defined as $\sum_{i,j} a_{ij}^2$. If $\boldsymbol{A}$ is binary, $\|\boldsymbol{A}\|_F^2 = |\boldsymbol{A}|$.

*Max-Planck Institute for Informatics, Saarbrücken, Germany. pauli.miettinen@mpi-inf.mpg.de

If $\boldsymbol{A}$ is a matrix and c is a scalar, $c+\boldsymbol{A}$ is a shorthand for increasing each element of $\boldsymbol{A}$ by c.

Let $\boldsymbol{B}$ be n-by-k and $\boldsymbol{C}$ be k-by-m binary matrices. Their *Boolean matrix product*, $\boldsymbol{B}\circ\boldsymbol{C}$, is the binary matrix $\boldsymbol{A}$ with $a_{ij} = \bigvee_{l=1}^{k} b_{il}c_{lj}$.

The *Boolean rank* of an n-by-m binary matrix $\boldsymbol{A}$, $\operatorname{rank}_B(\boldsymbol{A})$, is the least integer k such that there exists an n-by-k binary matrix $\boldsymbol{B}$ and a k-by-m binary matrix $\boldsymbol{C}$ for which $\boldsymbol{A} = \boldsymbol{B}\circ\boldsymbol{C}$. Matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ are *factor matrices* of $\boldsymbol{A}$, and the pair $(\boldsymbol{B},\boldsymbol{C})$ is the (approximate) rank-k *Boolean factorization* of $\boldsymbol{A}$.

A binary vector $\boldsymbol{a}$ is *dominated* by a binary vector $\boldsymbol{b}$ if $a_i \leq b_i$ for all i. The same terminology is extended to binary matrices. A Boolean factorization $(\boldsymbol{B},\boldsymbol{C})$ of a binary matrix $\boldsymbol{A}$ is dominated if $\boldsymbol{B}\circ\boldsymbol{C}$ is dominated by $\boldsymbol{A}$. Factorization $(\boldsymbol{B},\boldsymbol{C})$ of $\boldsymbol{A}$ *covers* a_{ij} if $(\boldsymbol{B}\circ\boldsymbol{C})_{ij} = 1$.

Finally, given a proposition P, in Iverson bracket notation $[P] = 1$ if P is true, and $[P] = 0$ otherwise.

3 The Problem

The Joint Subspace Boolean Matrix Factorization is an amalgam of two ideas: the Joint Subspace Matrix Factorization and Boolean Matrix Factorization. These two ideas are covered first, before giving the definition of the Joint Subspace Boolean Matrix Factorization.

3.1 Background on Joint Subspace Matrix Factorization.

The idea of Joint Subspace Matrix Factorization (JSMF) is to factorize two matrices with equal number of rows into three subspaces: one is shared between the two matrices, and two are specific to the matrices. The motivations for doing JSMF are varied, including the aim to separate the shared space from data-specific spaces and the aim of improving the factorization of one matrix with the auxiliary data of the other matrix. Formally, we can define the Joint Subspace Matrix Factorization as follows.

PROBLEM 3.1. (JSMF) *Given an n-by-m matrix $\boldsymbol{X}$, an n-by-l matrix $\boldsymbol{Y}$, and positive integers k, p, and q, find n-by-k matrix $\boldsymbol{W}$, n-by-p matrix $\boldsymbol{U}$, n-by-q matrix $\boldsymbol{V}$, $(k+p)$-by-m matrix $\boldsymbol{H}$, and $(k+q)$-by-l matrix $\boldsymbol{L}$ such that $\boldsymbol{X} \approx [\boldsymbol{W}\ \boldsymbol{U}]\boldsymbol{H}$ and $\boldsymbol{Y} \approx [\boldsymbol{W}\ \boldsymbol{V}]\boldsymbol{L}$.*

In Problem 3.1, matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the data matrices, matrix $\boldsymbol{W}$ defines the *shared subspace*, matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ define the *specific subspaces*, and matrices $\boldsymbol{H}$ and $\boldsymbol{L}$ are the *mixing matrices*. The definition of Problem 3.1 leaves open the precise error metric (loss function), as it is application-specific. The Frobenius norm is a common choice, however.

An important variation of JSMF is the Joint Subspace Nonnegative Matrix Factorization, JSNMF, proposed by Gupta et al. [4]. Along the lines of standard Nonnegative Matrix Factorization, JSNMF requires all involved matrices to be nonnegative. This gives 'parts-of-whole' representation of the data, possibly helping with interpretation and when applying the factorization to other problems. The particular application Gupta et al. [4] concentrate is the media retrieval based on tags. It is also claimed that JSNMF yields to sparse representation of the data.

Gupta et al. [4] also suggest weighting the loss function. If the two data matrices, $\boldsymbol{X}$ and $\boldsymbol{Y}$, have very different norms, the one with larger norm dominates the overall error. The normalized loss function is

(3.1)
$$\left\|\boldsymbol{Y}\right\|_F^2 \left\|\boldsymbol{X} - [\boldsymbol{W}\ \boldsymbol{U}]\boldsymbol{H}\right\|_F^2 + \left\|\boldsymbol{X}\right\|_F^2 \left\|\boldsymbol{Y} - [\boldsymbol{W}\ \boldsymbol{V}]\boldsymbol{L}\right\|_F^2 .$$

3.2 The Boolean Case.

Many data sets that could be used in JSMF are binary by nature, that is, they contain only 0s and 1s. For example, any data that records the presence (or absence) of variables in observations is binary. The tags of media files are a specific example, as tags usually do not have quantity attached to them. When data is binary, it is often natural to think it as a collection of sets, for example, each image is attached to a set of tags. But set arithmetics is different from normal arithmetics: if an element is in two sets, their union still has that element only once. In terms of matrix factorization, this idea is captured in the *Boolean Matrix Factorization*.

The Boolean Matrix Factorization (BMF) is like normal matrix factorization, except that all involved matrices (data and factors) are required to be binary and the Boolean matrix product is used. As the factors are binary, they can be interpet as sets, and the Boolean matrix multiplication corresponds to the set union of the factors.

Another benefit of Boolean Matrix Factorization is that it is show to yield sparse factorizations [9]. Indeed, the factor matrices can be considerably sparser than those obtained via Nonnegative Matrix Factorization.

3.3 Problem Definition.

We are now ready to define the main problem of this paper, the Joint Subspace Boolean Matrix Factorization (JSBMF) problem.

PROBLEM 3.2. (JSBMF) *Given an n-by-m binary matrix $\boldsymbol{X}$ and an n-by-l binary matrix $\boldsymbol{Y}$ and positive integers k, p, and q, find n-by-k binary matrix $\boldsymbol{W}$, n-by-p binary matrix $\boldsymbol{U}$, n-by-q binary matrix $\boldsymbol{V}$, $(k+p)$-by-m binary matrix $\boldsymbol{H}$, and $(k+q)$-by-l binary matrix $\boldsymbol{L}$ such that they minimize*

$$|\boldsymbol{X} - [\boldsymbol{W}\ \boldsymbol{U}]\circ\boldsymbol{H}| + |\boldsymbol{Y} - [\boldsymbol{W}\ \boldsymbol{V}]\circ\boldsymbol{L}| . \tag{3.2}$$

Equation (3.2) gives the unweighted reconstruction error. If, say, $|\boldsymbol{X}| \ll |\boldsymbol{Y}|$, this can yield unbalanced

results. Similar to (3.1), we can add weights so that the errors are proportional to the number of 1s in the matrices. This gives us

$$(3.3)\quad \left|\boldsymbol{Y}\right|\left|\boldsymbol{X}-[\boldsymbol{W}\ \boldsymbol{U}]\circ\boldsymbol{H}\right|+\left|\boldsymbol{X}\right|\left|\boldsymbol{Y}-[\boldsymbol{W}\ \boldsymbol{V}]\circ\boldsymbol{L}\right|\,.$$

3.4 Computational Complexity. The computational complexity of JSBMF does not differ from BMF, not at least in the case of lower bounds: setting $\boldsymbol{X}=\boldsymbol{Y}$ and $p=q=0$ the problem reduces back to the normal BMF, and therefore we can conclude that JSBMF is NP-hard even to approximate [10].

4 The Algorithms

We will present three different algorithms for JSBMF. The first could be considered more as a baseline method against which the other two are tested. The second is a relatively straight forward extension of an existing method for BMF, and the third algorithm is a modification of the second that can circumvent some of its potential problems. All these three algorithms need to solve the BMF problem. An adapted version of the `Asso` algorithm [10] is used for that purpose.

4.1 An Adapted `Asso` Algorithm and a Greedy Algorithm for Updating the Factors. To do the BMF, we employ the `Asso` algorithm of Miettinen et al. [10], but modified to work with element weights. To explain how this is obtained, we need to know how `Asso` works. Due to space constraints, the full `Asso` algorithm is not presented, however, but focus is on what will change. Full details of the algorithm can be found from [10]. Notice, however, that we use the modified `Asso` algorithm as a black box; any algorithm for BMF that accepts element-wise weights could be used instead.

The algorithm first creates a set of candidate (column) factors based on pairwise row association confidencies of the data (see [10] for more details). It then selects, greedily, one of the candidate factors to be the first column factor, and builds the corresponding row factor. This selection is done based on cover function, which is computed for each candidate factor and each column of the data matrix. If $\boldsymbol{a}$ is a column of the data matrix and $\boldsymbol{b}$ is a candidate factor, cover is defined as

$$(4.4)\quad \mathsf{cover}(\boldsymbol{a},\boldsymbol{b},c)=\sum_i\big(c(i)[a_i=1]-c(i)[a_i=0]\big)[b_i=1]\,,$$

where $c(i)$ is 0 if a_i is already covered by the decomposition, and $c(i)=1$ otherwise.

Initially $c(i)=1$ for all i. The selected candidate is the one that has highest cover value summed over all columns of the data matrix. The corresponding row factor has 1s for those columns where cover is positive. After the row and column factors are build, function c is updated to have zeros on those elements that are already covered.

Updating `Asso` to work with weights is straight forward. We just replace c with a weight function w that gives (positive) weight for each element that is not yet covered, and 0 for the covered elements. This is enough to make sure that `Asso` minimizes the weighted loss. The calling sequence of `Asso` is $\texttt{Asso}(\boldsymbol{X},k,\boldsymbol{N})$, where $\boldsymbol{X}$ is the data matrix, k is the rank of the decomposition, and $\boldsymbol{N}$ is a (nonnegative) matrix of same size than $\boldsymbol{X}$ that defines the element-wise weights.

The second algorithm we need is an algorithm to (greedily) update the factors. The algorithm, called `updateFactors` works as follows. Given $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$, the algorithm first considers each element of $\boldsymbol{W}$ and sees if 'flipping' the value (i.e. setting $w_{ij}=1-w_{ij}$) would reduce the (possibly weighted) cost. After it has iterated over $\boldsymbol{W}$, it iterates over matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, and then over matrices $\boldsymbol{H}$ and $\boldsymbol{L}$. After this, it returns the updated factors. An important property of the algorithm is that it is guaranteed to either return the original factors or to reduce the error by at least 1.

The `updateFactors` method is very simple method. It might seem that it will be overly expensive to use, but in fact this was not the case in the experiments. The time complexity is $O(nml\max\{n,m,l\}^2)$ at worst case, but in practice it is faster as we can speed up the Boolean matrix multiplication by noticing that we only need to compute the sum defining the value of a matrix element up to the point where we see the first 1. Also we can parallelize the computation: the rows of $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{V}$, and the columns of $\boldsymbol{H}$ and $\boldsymbol{L}$ can be updated simultaneously as they do not interfere with each other. We can also update $\boldsymbol{V}$ and $\boldsymbol{U}$, and $\boldsymbol{H}$ and $\boldsymbol{L}$ simultaneously.

4.2 The `ConcatAsso` Algorithm. The simplest way to solve a joint subspace factorization is by concatenating the two data matrices into one and decomposing that matrix using existing tools. This is exactly what `ConcatAsso` does: given $\boldsymbol{X}$ and $\boldsymbol{Y}$, it finds the Boolean factorization of $[\boldsymbol{X}\ \boldsymbol{Y}]$. The rank of the BMF decomposition is set to $k+p+q$, such that the factorization contains n-by-$(k+p+q)$ binary matrix $\boldsymbol{B}$ and $(k+p+q)$-by-$(m+l)$ binary matrix $\boldsymbol{C}$. The factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ are split into the JSBMF factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$ as follows.

Consider $\boldsymbol{b}$, a column of $\boldsymbol{B}$, and its corresponding row of $\boldsymbol{C}$, $\boldsymbol{c}$. If $\boldsymbol{c}$ has 1s only on first m columns (corresponding to the columns of $\boldsymbol{X}$), vector $\boldsymbol{b}$ is a column of $\boldsymbol{U}$ and $\boldsymbol{c}$ (truncated to proper length) is added to $\boldsymbol{H}$; if $\boldsymbol{c}$ has 1s only on columns corresponding to the columns

Algorithm 1 The baseline algorithm for JSBMF.

Input: Binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, parameters k, p, and q.
Output: Binary factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$.

```
function ConcatAsso(X, Y, k, p, q)
    (B, C) ← Asso([X Y], k + p + q)
    (W, U, V, H, L) ← splitFactors(B, C)
    repeat
        (W, U, V, H, L) ← updateFactors(X, Y, W, U, V, H, L)
    until converged
    return W, U, V, H, L
```

Algorithm 2 A greedy algorithm for JSBMF.

Input: Binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, parameters k, p, and q.
Output: Binary factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$.

```
function SharedAsso(X, Y, k, p, q)
    (W, [H_W L_W]) ← Asso([X Y], k)
    (U, H_U) ← Asso(X, p, 1 − W ∘ H_W)
    (V, L_V) ← Asso(Y, q, 1 − W ∘ L_W)
    H ← [H_W^T H_U^T]^T; L ← [L_W^T L_V^T]^T
    repeat
        (W, U, V, H, L) ← updateFactors(X, Y, W, U, V, H, L)
    until converged
    return W, U, V, H, L
```

of $\boldsymbol{Y}$, vector $\boldsymbol{b}$ is a column of $\boldsymbol{V}$ and truncated $\boldsymbol{c}$ is added to $\boldsymbol{L}$; otherwise $\boldsymbol{b}$ is a column of $\boldsymbol{W}$ (we assume $\boldsymbol{c}$ is not empty) and $\boldsymbol{c}$ is split into two vectors that are added to $\boldsymbol{H}$ and $\boldsymbol{L}$. The overview of `ConcatAsso` algorithm is given in Algorithm 1 where the splitting process is referred to as `splitFactors`.

For the sake of clarity the pseudocode of `ConcatAsso` (and all upcoming pseudocodes) is presented without the weighting of (3.3). Applying it is, however, straight forward: the weight parameter for `Asso` has to be changed, and `updateFactors` has to use weighting.

The `ConcatAsso` algorithm will not necessarily yield to $\boldsymbol{W}$ with k columns or $\boldsymbol{V}$ with p columns (or $\boldsymbol{U}$ with q columns); it can be, for example, that one of these matrices is completely empty. Therefore, in the strict sense, `ConcatAsso` does not solve the Problem 3.2. Instead, it has more power as it can define the number of factors such that they minimize the reconstruction error. As such, it is to be assumed that it will produce the smalles reconstruction error of the proposed methods (this, indeed, is the case – see Section 6).

The time complexity of `ConcatAsso` is dominated by that of `updateFactors`, and by the number of times it has to be called. As the `updateFactors` algorithm reduces the error in each iteration by at least 1, the algorithm will converge in at most $n(m + l)$ steps. In practice the iterative update converges very fast, often in less than 5 iterations.

4.3 The `SharedAsso` Algorithm. The first algorithm to actually solve Problem 3.2 is called `SharedAsso`. Like `ConcatAsso` it is based on the `Asso` algorithm, but with slightly more complex implementation. The algorithm will work in three phases. In the first phase it finds the shared factors by calling `Asso` with $[\boldsymbol{X}\ \boldsymbol{Y}]$ and with k factors (unlike `ConcatAsso` that used $k + p + q$ factors). The result of `Asso` provides the initial matrices $\boldsymbol{W}$, $\boldsymbol{H_W}$, and $\boldsymbol{L_W}$ (the latter two matrices are the first k rows of $\boldsymbol{H}$ and $\boldsymbol{L}$, respectively).

In the second phase it calls `Asso` twice to produce the initial specific factors for $\boldsymbol{X}$ and $\boldsymbol{Y}$. But to take the already-found shared factors into account, it sets zero weight for those elements of $\boldsymbol{X}$ and $\boldsymbol{Y}$ that are already covered by the shared factors (i.e. elements that 1 in $\boldsymbol{W} \circ \boldsymbol{H_W}$ or $\boldsymbol{W} \circ \boldsymbol{L_W}$).

At the begin of the third phase, `SharedAsso` has initial versions of $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$. It then calls `updateFactors` to obtain the final versions. The pseudocode of `SharedAsso` is provided in Algorithm 2.

As with `ConcatAsso`, the time complexity of `SharedAsso` is dominated by that of `updateFactors` and the number of iterative updates done.

4.4 The `Dominated` Algorithm. The Boolean decomposition has the property that if one pair of corresponding row and column factors cover some 0 of the data matrix, that error cannot be fixed without changing those factors. Because of this, overly greedy initial selection can yield bad end results.

These thoughts serve as the motivation for our third algorithm, the `Dominated` algorithm. It has one major difference to `SharedAsso`: the way the initial matrix $\boldsymbol{W}$ is selected. Instead of using `Asso`, `Dominated`, as the name indicates, finds dominated shared factors (i.e. factors that cover only 1s in the data).

To do this, it uses the greedy Set Cover (or rather, Max k-Cover) algorithm. In short, the idea is to reduce the BMF problem to the Set Cover problem in such a way that the 1s in the data matrix correspond to the elements in the ground set, and the sets in the set system are all possible dominated factors [1, 9]. One can then apply the standard greedy heuristic [7] to this set system to obtain $\mathrm{e}/(\mathrm{e}-1)$ approximation for the Max k-Cover problem. This translates into $\mathrm{e}/(\mathrm{e}-1)$ approximation of the dominated BMF [9].

The problem with this approach is that if the data matrix is not sparse, the number of sets in the set system grows exponentially, yielding exponential-time algorithm. Miettinen [9] characterized the sparsity properties required to keep the algorithm polynomial-

time, but we cannot expect all our data sets to fulfill those requirements.

To overcome this problem we approximate the set system itself when needed. Intuitively, the largest sets matter most and we can omit the small ones. But how to effectively find the largest ones? The answer is to use frequent itemset mining. A large set in the set system corresponds to a factor that covers many 1s in the data without covering any 0s (i.e. is dominated). But such factors are exactly the large, monochromatic submatrices, i.e. *tiles* [2]. Assuming there is no minimum frequency threshold for the itemsets, this gives us exactly the same approximation result.

PROPOSITION 4.1. *Using the Maximum k-tiling [2], we can obtain an $\mathrm{e}/(\mathrm{e}-1)$ approximation of the dominated BMF.*

Proof. A maximum k-tiling of a matrix is a set of k tiles (all-1s submatrices) that together cover as much of the 1s of the matrix as possible. Each tile defines corresponding pair of row and column factors as each tile can be expressed (uniquely) as an outer product of two binary vectors. Geerts, Goethals, and Mielikäinen [2] proved that maximum k-tiling can be approximated within $\mathrm{e}/(\mathrm{e}-1)$, from which the claim follows.

So far we have not been able to overcome the problem of too many sets (the proof of [2] requires an access to an oracle), we have just changed the terminology from sets to tiles. The approach we use here is to only use tiles induced by closed itemsets. When using this collection, the algorithm is no more guaranteed to admit any approximation factor, but the number of sets is reduced considerably. Furthermore, we require that the itemsets have items from both matrices.

To sum up, the `Dominated` algorithm works as follows. First, it finds dominated shared factors for the initial versions of $\boldsymbol{W}$, $\boldsymbol{H_W}$, and $\boldsymbol{L_W}$. Then it proceeds as `SharedAsso`: it calls `Asso` with the weight of the already-covered elements set to 0, and with those results, starts the iterative update phase `updateFactors`. The pseudocode is given in Algorithm 3.

Notice that the greedy Max k-cover algorithm lends itself nicely to element-wise weights. The algorithm works as well with the elements having weights, and the requirement for the dominated factors means that we do not have to worry about the costs of covering 0s.

The time complexity of `updateFactors` is in practice (and in theory) dominated by the tiling which, depending on the minimum frequency parameter f, can take exponential time. Other than that, its behaviour is akin to the two other algorithms, `ConcatAsso` and `SharedAsso`.

Algorithm 3 A tiling-based algorithm for JSBMF.

Input: Binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, parameters k, p, and q, minimum frequency f.
Output: Binary factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$.

1: **function** `Dominated`$(\boldsymbol{X}, \boldsymbol{Y}, k, p, q, f)$
2: $(\boldsymbol{W}, [\boldsymbol{H_W}\ \boldsymbol{L_W}]) \leftarrow$ `tiling`$([\boldsymbol{X}\ \boldsymbol{Y}], k, f)$
3: $(\boldsymbol{U}, \boldsymbol{H_U}) \leftarrow$ `Asso`$(\boldsymbol{X}, p, 1 - \boldsymbol{W} \circ \boldsymbol{H_W})$
4: $(\boldsymbol{V}, \boldsymbol{L_V}) \leftarrow$ `Asso`$(\boldsymbol{Y}, q, 1 - \boldsymbol{W} \circ \boldsymbol{L_W})$
5: $\boldsymbol{H} \leftarrow [\boldsymbol{H_W^T}\ \boldsymbol{H_U^T}]^T$; $\boldsymbol{L} \leftarrow [\boldsymbol{L_W^T}\ \boldsymbol{L_V^T}]^T$
6: **repeat**
7: $(\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H}, \boldsymbol{L}) \leftarrow$ `updateFactors`$(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H}, \boldsymbol{L})$
8: **until** converged
9: **return** $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H}, \boldsymbol{L}$

5 Selecting the Correct Approximation Ranks

An important problem with any matrix decomposition method is the selection of the rank of the approximation. The problem is even more pronounced with joint matrix factorizations, where the user has to select not just one, but three ranks.

To solve the problem of selecting the ranks, we apply the Minimum Description Length (MDL) principle. The MDL principle says that the correct ranks are those that allow us to represent the data with fewest bits.[1] To use the MDL principle, we must first select some ranks, compute the factorization, and then compute the encoding length of the factorization. After repeating this process with different ranks, we can select the combination of ranks that gives the shortest encoding.

To apply the MDL principle, we will encode the factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$, and the error introduced by the factorization. The error is represented as matrices $\boldsymbol{E_X} = \boldsymbol{X} \otimes \big([\boldsymbol{W}\ \boldsymbol{U}] \circ \boldsymbol{H}\big)$ and $\boldsymbol{E_Y} = \boldsymbol{Y} \otimes \big([\boldsymbol{W}\ \boldsymbol{V}] \circ \boldsymbol{L}\big)$, where $\otimes$ is the element-wise exclusive-or operator. With the knowledge of factors and error, we can reconstruct $\boldsymbol{X}$ and $\boldsymbol{Y}$ exactly.

The intuition of using MDL is that expressing structure with factors and noise with noise matrices takes less bits than if factors are expressed with noise matrices or noise with factors. MDL is also well-suited for JSBMF, as it selects the individual ranks of the subspaces. Let $\boldsymbol{u}$ be a specific factor for $\boldsymbol{X}$ and $\boldsymbol{v}$ be a specific factor of $\boldsymbol{Y}$. If we remove $\boldsymbol{u}$ and $\boldsymbol{v}$ and replace them with a single shared factor $\boldsymbol{w}$ we (typically) save on encoding length of the factor matrices. Depending how similar $\boldsymbol{u}$ and $\boldsymbol{v}$ were, this saving can carry on to the total encoding length, or be nullified by the increase of encoding length for error matrices. In the former case, we consider the replacement of $\boldsymbol{u}$ and $\boldsymbol{v}$ with $\boldsymbol{w}$ beneficial even if the error is slightly increased. Notice that if we

[1] For more information about MDL, see [3].

let k, p, and q vary, but fix $k+p$ and $k+q$, and aim just to minimize the error, letting $k=0$ yields always the best solution.

To compute the encoding length, we will adapt a recently-proposed method of using MDL for standard Boolean matrix factorization [11].

First we need to encode the factor matrices. In MDL parlance, this corresponds to the encoding of the model (or hypothesis). We start by encoding the dimensions n, m, and l using Elias-Delta coding [3], requiring $\log(x) + 2\log\big(\log(x)+1\big)+1$ bits to encode integer x. We then encode the ranks k, p, and q. We do not wish to introduce bias for small ranks by using Elias-Delta here, so instead we encode them using fixed number of bits. This works as we can bound the ranks from given n, m, and l. To encode k, we need $L(k) = \log(\min\{n,m,l\})$ bits (if $k = \min\{n,m,l\}$, at least one of the data matrices can be encoded trivially using identity matrix as the other factor). With k known, we can further bound p and q:

$$L(p) = \log(\min\{n, m-k\}) \text{ and } L(q) = \log(\min\{n, l-k\}).$$

Following [11], we encode each factor separately. This means that we encode columns of $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ and rows of $\boldsymbol{H}$ and $\boldsymbol{L}$ independently. We will explain how to encode $\boldsymbol{W}$; other factor matrices are encoded analogously. Each column of $\boldsymbol{W}$ is encoded using the optimal prefix code. This can be computed when one knows the fraction of 1s in the column, $p_1^{\boldsymbol{w}_i} = |\boldsymbol{w}_i|/n$. Value $p_1^{\boldsymbol{w}_i}$ takes $\log n$ bits to encode. The optimal prefix code lengths for 1 and 0 in a column $\boldsymbol{w}_i$ are then

$$l_1(\boldsymbol{w}_i) = -\log(p_1^{\boldsymbol{w}_i}) \quad \text{and} \quad l_0(\boldsymbol{w}_i) = -\log(1-p_1^{\boldsymbol{w}_i}).$$

The number of bits we need to encode the matrix $\boldsymbol{W}$ is

$$L(\boldsymbol{W}) = k\log n + \sum_{i=1}^{k}\big(|\boldsymbol{w}_i|\, l_1(\boldsymbol{w}_i) + (n-|\boldsymbol{w}_i|)l_0(\boldsymbol{w}_i)\big). \tag{5.5}$$

Encoding the remaining factor matrices using analogous process finishes the model part. What remains is the error matrices $\boldsymbol{E_X}$ and $\boldsymbol{E_Y}$, i.e. the data given the model. To encode the error, we use the Typed XOR model from [11] (for motivation behind the model, and some alternative models, see [11]). This approach is based on idea of first dividing the error to false positives (i.e. those elements that are 0 in the data but are represented as 1) and false negatives (that are 1 in the data but are represented as 0). For $\boldsymbol{E_X}$, we denote these two error types by $\boldsymbol{E}_{\boldsymbol{X}}^+$ and $\boldsymbol{E}_{\boldsymbol{X}}^-$, respectively, yielding $\boldsymbol{E_X} = \boldsymbol{E}_{\boldsymbol{X}}^+ + \boldsymbol{E}_{\boldsymbol{X}}^-$.

Let $\widetilde{\boldsymbol{X}} = [\boldsymbol{W}\ \boldsymbol{U}] \circ \boldsymbol{H}$. We define the probability of 1s in $\boldsymbol{E}_{\boldsymbol{X}}^+$ to be $p_1^+ = \big|\boldsymbol{E}_{\boldsymbol{X}}^+\big|/(nm - |\widetilde{\boldsymbol{X}}|)$ and the probability of 1s in $\boldsymbol{E}_{\boldsymbol{X}}^-$ to be $p_1^- = \big|\boldsymbol{E}_{\boldsymbol{X}}^-\big|/|\widetilde{\boldsymbol{X}}|$. With these probabilities we can again make optimal prefix codes for 1s and 0s in $\boldsymbol{E}_{\boldsymbol{X}}^+$ and $\boldsymbol{E}_{\boldsymbol{X}}^-$. For $\boldsymbol{E}_{\boldsymbol{X}}^+$ these have length $l_1^+ = -\log p_1^+$ and $l_0^- = -\log(1-p_1^+)$, respectively. Prefix code lengths l_1^- and l_0^- are computed analogously. We now have

$$L(\boldsymbol{E_X}) = L(\boldsymbol{E}_{\boldsymbol{X}}^+) + L(\boldsymbol{E}_{\boldsymbol{X}}^-) \tag{5.6}$$

with

$$\begin{aligned} L(\boldsymbol{E}_{\boldsymbol{X}}^+) &= \log(nm - |\widetilde{\boldsymbol{X}}|) + \big|\boldsymbol{E}_{\boldsymbol{X}}^+\big|\, l_1^+ \\ &\quad + (nm - |\widetilde{\boldsymbol{X}}| - \big|\boldsymbol{E}_{\boldsymbol{X}}^+\big|)l_0^+ \\ L(\boldsymbol{E}_{\boldsymbol{X}}^-) &= \log|\widetilde{\boldsymbol{X}}| + \big|\boldsymbol{E}_{\boldsymbol{X}}^-\big|\, l_1^- + (|\widetilde{\boldsymbol{X}}| - \big|\boldsymbol{E}_{\boldsymbol{X}}^-\big|)l_0^-. \end{aligned}$$

The length for encoding $\boldsymbol{E_Y}$ is computed analogously, finishing our encoding length computation.

6 Experimental Evaluation

The proposed methods were compared using both synthetic and real-world data. The effectiveness of the MDL rank selection was also studied with both types of data. In addition to the Boolean methods proposed here, the `JSNMF` algorithm proposed by Gupta et al. [4] was used.

Comparing Boolean methods to `JSNMF` is not straight forward. When $\boldsymbol{X}$ and $\boldsymbol{Y}$ are binary the normalization constants in (3.1) become just the number of 1s in the respective matrices, i.e. the weighting is the same as used with the Boolean methods. But as the factor matrices are allowed to be non-negative rather than binary, the errors are not computed equivalently. As the element-wise errors are squared, all residual errors less than 1 are considered less than their face value; this generally favors dense factors that yield small errors in most (if not all) elements. As the Boolean methods cannot have errors smaller than 1, using Frobenius distance generally places them in a disadvantage. On the other hand, if we use sum of absolute differences, the small error made by `JSNMF` yields high overall error – but this approach is unfair to `JSNMF`.

We resolve this problem by reporting, when applicable, both types of error for `JSNMF`. When we report the error in the sum of absolute distances, the method is referred as $\texttt{JSNMF}_\texttt{B}$, and when we report the error using the squared residual error, we use $\texttt{JSNMF}_\texttt{F}$. In both cases the underlying algorithm is the same, just the error metrics changes.

As `JSNMF` is based on iterative updates form a random starting point, all results are the best of 13 restarts with identical parameters. The maximum number of iterations was set to 300.

The various parts of the algorithms were implemented using Matlab, C, and Python. For mining the

closed itemsets we used Christian Borgelt's implementation[2] of the FPgrowth algorithm [5].

6.1 Synthetic Data. The purpose of the experiments with the synthetic data is to study, in a controlled manner, the effects various data characteristics have to the algorithms. These characteristics are (*i*) noise level; (*ii*) number of shared factors; (*iii*) number of specific factors; and (*iv*) density of the factors. In addition, we also varied l, the number of columns in $\boldsymbol{Y}$: it was set to either $l = 40$, representing a case with a narrow data matrix, or to $l = 200$, representing a case with a more square data matrix. The number of rows in $\boldsymbol{X}$ and $\boldsymbol{Y}$ was always 150 and the number of columns in $\boldsymbol{X}$ was always 110.

To create the synthetic data we created random factor matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{H}$, and $\boldsymbol{L}$ for each combination of parameters. The factor matrices were multiplied to form $\boldsymbol{X}$ and $\boldsymbol{Y}$, after which the noise was added. This process was repeated to create 10 random data matrix pairs for each parameter combination. The default values for the parameters were: noise level 5% (with respect to the number of 1s in the data), number of shared factors $k = 10$, number of specific factors $p = q = 10$, and density of factor matrices 0.05.

In the following figures, Boolean methods are compared to $\texttt{JSNMF}_\texttt{F}$ when the Boolean methods used weighted loss (i.e. when all methods used the same weighting scheme). Notice that the error of $\texttt{JSNMF}_\texttt{F}$ is the weighted squared Frobenius distance, not the (weighted) sum of absolute distances. That error was also computed ($\texttt{JSNMF}_\texttt{B}$), but the results were considerably worse than those of the other methods, and are omitted. In the figures, all points are averages over the 10 random matrices (5 in Fig. 5) and the width of the error bars is twice the standard deviation.

Noise. Figure 1 presents the results with varying noise level. Here $l = 200$ and the noise level is reported with respect to the number of 1s in the data matrices.

From Fig. 1(a) (unweighted loss) we can see that, rather unsurprisingly, the error of the methods increases as the noise level increases. Furthermore, the increase is almost linear (notice that the x-axis is not linear). All three methods are close to each other, with the order being, as expected, `SharedAsso` (the worst), `Dominated`, and `ConcatAsso` (the best).

Using the weighted loss (Fig. 1(b)) does not change the ordering of the Boolean methods. But somewhat surprisingly, $\texttt{JSNMF}_\texttt{F}$ is worse than `Dominated` or `ConcatAsso` with smaller amounts of noise. Apparently here the Boolean methods can use the power of the Boolean arithmetic while `JSNMF` cannot benefit form its larger expressive power.

Number of Shared Factors. Figure 2 presents the error when the number of shared factors increases. This also yields increase in the reconstruction error, as is to be expected: the more there are latent factors, the more complex combinations of them appear in the data, making the decomposition harder. In all cases the order of the Boolean methods is the same as it was in Fig. 1.

In Fig. 2(a) we can see the results when $\boldsymbol{Y}$ is narrow, that is, has only 40 columns. Here, when $k < 20$, all methods have very small variance and results close to each other; with $k = 20$, though, $\texttt{JSNMF}_\texttt{F}$ is clearly the best, and all methods have much higher standard deviation.

With $\boldsymbol{Y}$ having 200 columns (Fig. 2(b)), the results are slightly different. With $k = 5, 10$, `Dominated` is better than $\texttt{JSNMF}_\texttt{F}$, and still with $k = 20$, `ConcatAsso` is the best.

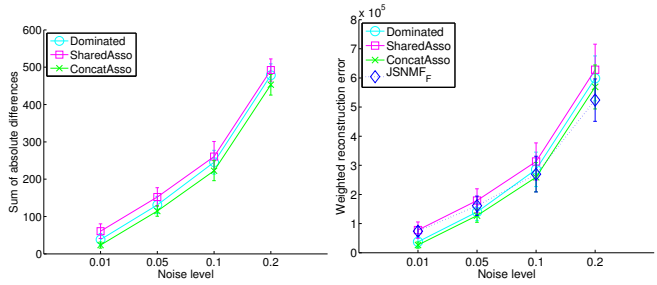
Number of Specific Factors. The results for varying the number of specific factors (Fig. 3) mostly follow those presented above. Again, with low values of p and q, $\texttt{JSNMF}_\texttt{F}$ is slightly better than `SharedAsso`, but slightly worse than `Dominated` and `ConcatAsso`. With $p = 10, q = 30$ it is in par with `ConcatAsso`. Note, however, that with weighted errors (Fig. 3(b)) and $p = 10, q = 30$, `Dominated` is just slightly worse than `SharedAsso`. The probable reason for this is discussed below.

Density of the Factor Matrices. The results of Fig. 4 probably explain some of the phenomena we have seen above. First, in weighted and unweighted case we can see that as the density increases, `Dominated` becomes significantly worse. This not surprising, as higher minimum support threshold for closed itemset mining was used for denser matrices. In other experiments, and here with factor matrix density 0.05, no minimum support threshold was used; with factor matrix density 0.2, the resulting data matrix was so dense[3] that minimum frequency threshold of 0.6 was used in order to make the algorithm run in competitive time. This clearly had the expected adverse effect on the results.

On the other hand, from Fig. 4 we also see that $\texttt{JSNMF}_\texttt{F}$ is, for the first time, clearly the best method. This probably explains why it is doing better with higher number of shared or specific factors: as the overall number of factors is increased, the expected density of the data also increases, and $\texttt{JSNMF}_\texttt{F}$ seems to be least affected with that.

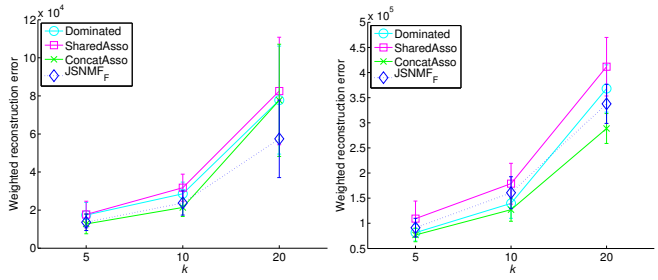
[2] `http://www.borgelt.net/fpgrowth.html`

[3] The expected density of the data matrices is $1 - (1 - d^2)^{k+p}$, where d is the density, i.e. with $d = 0.2$ and $k = p = 10$, the expected density is 0.56.



(a) (b)

Figure 1: Reconstruction error with respect to noise. (a) Unweighted loss. (b) Weighted loss.



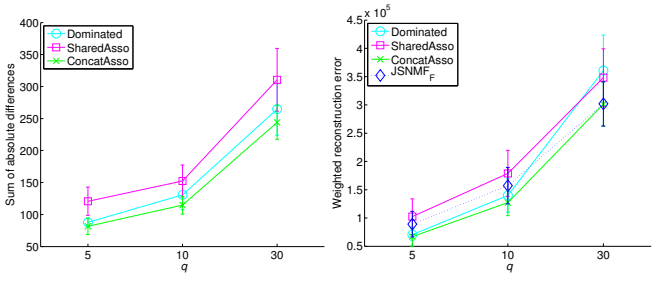
(a) (b)

Figure 2: Weighted reconstruction error with respect to the number of shared factors. (a) $\boldsymbol{Y}$ has 40 columns. (b) $\boldsymbol{Y}$ has 200 columns.

Selecting the Rank. The MDL method for selecting the ranks was tested with data for varying k, varying p and q, and varying noise level. Only the $l = 200$ version was used. In the first two cases, we only tried to estimate the varied parameter (k or $p = q$), while the other parameter(s) were kept constant. With varying noise levels, all parameters were varied. The results for varying k and noise level are presented in Fig. 5. The results for varying p and q were similar to the results for varying k.

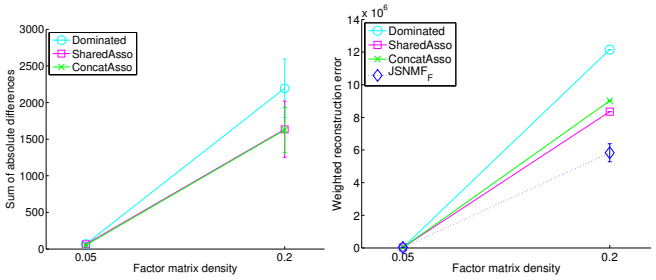
To estimate k, the algorithms were run with k taking values from 1 to 40, and the value of k giving the smallest encoding length was selected. This was repeated for the five random copies of the synthetic data, and the results in Fig. 5(a) are the averages over these matrices. Generally all methods work reasonably well, though `SharedAsso` tends to overestimate, and, with true $k = 20$, the other two underestimate slightly.

With varying noise level, all parameters were estimated. True parameters were $k = p = q = 10$, and the tried values were $2, 4, 6, \ldots, 40$. In total 400 parameter configurations were tried as p and q were fixed equal. Again, the results are averages over five random data with identical parameters. The results, in Fig. 5(b),



(a) (b)

Figure 3: Reconstruction error with respect to the number of specific factors. (a) Unweighted loss. (b) Weighted loss. x-axis shows q, the number of specific factors to $\boldsymbol{Y}$. The number of specific factors to $\boldsymbol{X}$, p, was equal to q except with $q = 30$, when $p = 10$ was used.



(a) (b)

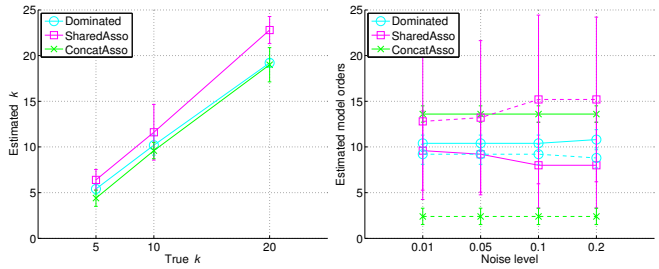
Figure 4: Reconstruction error with respect to factor matrix density. (a) Unweighted loss. (b) Weighted loss.

show clearly that `Dominated` is the best method in this experiment. It returns almost perfect answer (very slightly overestimating k and underestimating p and q). `SharedAsso` gives initially a good estimate to k, but starts to underestimate as noise level increases, and constantly overestimates p and q. In addition, `SharedAsso` suffers from very large deviation within noise levels. Finally, `ConcatAsso` is not affected by noise, but constantly overestimates k and underestimates p and q.

Discussion. The synthetic experiments mostly agree with the intuition: of the Boolean methods the `ConcatAsso` algorithm is the best, but with only small margin to the `Dominated` algorithm in most cases. The `Dominated` algorithm then is typically somewhat better than `SharedAsso`, except when the data is more dense.

Perhaps somewhat surprisingly, `JSNMF` is usually no better than the Boolean methods. Furthermore, `JSNMF` returns the densest factor matrices (this is studied more below) and was almost always the slowest method.

When estimating the ranks, `Dominated` is the best performer. Its performance also establishes that the proposed MDL method works.



(a) (b)

Figure 5: Estimated reconstruction ranks. (a) Estimated k for different true values of k. (b) Estimated ranks for true $k = p = q = 10$ and varying noise. Solid lines represent estimated k and dashed lines represent estimated p and q.

6.2 Real-World Data. The algorithms were also studied using five real-world data sets of varying characteristics. The main focus is again on the reconstruction error, but also the density of the factor matrices was studied. All data sets come with row and/or column labels, and we will also provide some examples on the found factors. But as the space constraints render all such studies anecdotal at the best, the focus will be on the numerical results.

The five data sets used are the following. The News data contains an excerpt of the 20 Newsgroups data[4]. Matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ contain posts from newsgroups `sci.space` and `soc.religion.christian`, respectively. Both matrices have 100 columns (documents) and the rows are 800 most common terms (excluding the stop words). As the data is binary, only the presence of the term is recorded, not its frequency. The DBLP[5] data contains information about authors that have published in 19 conferences and their co-authors. Here, rows correspond to the authors and columns of $\boldsymbol{X}$ are the conferences (i.e. $\boldsymbol{X}$ has 19 columns), while the columns of $\boldsymbol{Y}$ represent the same authors as the rows, and therefore $\boldsymbol{Y}$ is symmetric. The data has 2345 authors.

The Mammals data is a subset of the European Mammal Atlas[6] [12]. The rows correspond to spatial locations in Europe, and matrix $\boldsymbol{X}$ records the presence of carnivorous mammal species while matrix $\boldsymbol{Y}$ records the presence of herbivorous species. There are 2670 spatial locations, 40 carnivorous species, and 51 herbivorous species. The Web data[7] contains terms from the web pages of US Universities' CS departments. The rows correspond to the pages. The columns of $\boldsymbol{X}$ correspond to the terms found in the body text of the page and the columns of $\boldsymbol{Y}$ correspond to the terms found in links pointing to that page. There are 1051 pages, 1798 terms in the body text, and 438 terms in the links.

Finally, the Tags data is gathered from the LabelMe[8] [13] and Flickr[9] databases. The rows correspond to 119 tags associated to images, and the columns correspond to the images. The data has 735 images from LabelMe and 4159 images from Flickr.

Reconstruction Error. We only give the results with weighted loss functions (Table 1); the unweighted results were analogous. The two JSBMF methods, `Dominated` and `SharedAsso`, are mostly very close to each other, but unlike with synthetic data, here `SharedAsso` is often somewhat better. Part of this is probably due to the fact that `Dominated` had to use 5% minimum support with the Web data and 10% minimum support with the Mammals data. Nevertheless, the differences between the best and worst Boolean method (including `ConcatAsso`) are typically small, peaking at 15% with the Tags data.

In all other data sets except the DBLP data there is a pair of parameter configurations that keep the total number of factors used to explain each matrix constant (the first and the last row of each data sets' results). This allows us to study the effects of increasing the fraction of shared factors. One could assume that this decreases the reconstruction error slightly, as the shared factors cannot cover the specific parts of data that well.

Indeed, this seems to be the case. Interestingly, though, with News and Web data, increasing the total number of factors per matrix from 16 to 20 did not improve `ConcatAsso`'s results, and moreover, it even increased the error of `Dominated`.

Comparing the results of the Boolean methods to $\texttt{JSNMF}_\texttt{F}$ and $\texttt{JSNMF}_\texttt{B}$ we see that even if the Boolean methods were on a par with $\texttt{JSNMF}_\texttt{F}$ in the synthetic experiments, with real-world data $\texttt{JSNMF}_\texttt{F}$ obtains smaller reconstruction errors (the sole exception being the first row of Tags data, where `ConcatAsso` is better), whereas in all cases, $\texttt{JSNMF}_\texttt{B}$ is much worse than any other method (as expected). That $\texttt{JSNMF}_\texttt{F}$ is better than the Boolean methods is also in accordance with the results from the previous studies where Boolean and non-negative methods are compared (e.g. [10]). Yet, with the exception of the Mammals data, the difference is modest. The behaviour with the Mammals data is probably explained by the fact that it is the most dense of the tried data sets.

[4] `http://people.csail.mit.edu/jrennie/20Newsgroups/`
[5] `http://www.informatik.uni-trier.de/~ley/db/`
[6] `http://www.european-mammals.org`
[7] `http://www.cs.cmu.edu/afs/cs/project/theo-11/www/wwkb/`
[8] `http://labelme.csail.mit.edu/`
[9] `http://www.flickr.com/`

Table 1: Weighted reconstruction errors for real-world data. $\texttt{JSNMF}_\texttt{F}$ uses squared Frobenius distance, all others use sum of absolute distances. Values are scaled by 10^6.

Dataset	(k, p, q)	Dominated	SharedAsso	ConcatAsso	$\texttt{JSNMF}_\texttt{B}$	$\texttt{JSNMF}_\texttt{F}$
News	$(4, 16, 16)$	9.57	9.27	9.25	18.06	7.40
	$(8, 8, 8)$	11.62	11.03	11.02	20.96	8.87
	$(16, 4, 4)$	12.61	11.02	11.02	21.18	8.88
DBLP	$(4, 8, 20)$	288.76	265.90	250.16	496.19	221.27
	$(8, 4, 16)$	316.19	268.44	253.48	522.15	225.29
Mammals	$(4, 8, 8)$	196.79	183.13	174.11	294.85	117.16
	$(8, 4, 4)$	217.11	206.17	205.36	362.76	144.88
Web	$(4, 16, 16)$	306.73	303.83	297.66	560.00	260.08
	$(8, 8, 8)$	326.00	315.74	312.04	576.43	273.51
	$(16, 4, 4)$	339.78	312.79	312.04	593.77	271.24
Tags	$(4, 16, 16)$	161.12	157.03	121.19	287.54	135.59
	$(8, 8, 8)$	176.13	179.32	156.11	311.02	150.99
	$(16, 4, 4)$	166.84	162.60	156.11	299.57	142.15

Density of the Factors. In addition to the reconstruction error, the density (or sparsity) of the factor matrices is an important characteristic in many applications. The Boolean methods generally produce very sparse factors.

The matrix decomposition methods based on the standard linear algebra, on the other hand, tend to produce very dense factors, at least if no regularizes are used to control the density. We are not aware of any joint subspace matrix factorization algorithm that would use regularizes, and therefore we used JSNMF also in these experiments. The results are presented in Table 2.

As can be readily seen from Table 2, JSNMF produces much denser factors than any of the Boolean methods, which tend to produce very similar densities with each other. Unsurprisingly, SharedAsso produces almost always denser factors than Dominated.

Selecting the Ranks. The description lengths for the computed factorizations can be found in Table 3. The purpose of this experiment was not to find the optimum parameter combination for each data, as that would require very extensive computations; rather, the goal is to study if the selected parameter combinations give rise to any special phenomena. Thus the code lengths themselves are less interesting than their relations to each other.

The first notable phenomenon is that all values are close to each other within same data set. In the case of News, Mammals, and Web, SharedAsso and Dominated agree with the parameter combination. Excluding the Mammals data, ConcatAsso agrees with either one of the two.

Studying the trends within data, we notice that Mammals seems to benefit from moving factors to shared subspace, even if this increased the error (and average density). Similar thing happens with Tags and Dominated, where moving from $(4, 16, 16)$ to $(8, 8, 8)$ decreases the MDL score. This shows how the MDL principle works with JSBMF: even if moving factor matrices from specific subspaces to shared subspace can never improve the error (if the total number of factors is reduced), it can improve the MDL score by being easier to encode.

Interpertability of the Results. One of the main reasons behind using the Boolean methods is the interpretability of the results [10]. The iterpretability of plain BMF is studied previously (see e.g. [10]), and therefore we concentrate here on the specific properties of JSBMF.

Because of space constraints, we only report results with the News and Tags datasets. For News, the algorithm was Dominated without weighting (as the matrices are of the same size and of similar density, weighting would have not changed the results much) and the parameters were $k = p = q = 8$. The shared factors in $\boldsymbol{W}$ contain terms that appear in both newsgroups. This can be easily seen from the results. The factors in $\boldsymbol{W}$ are rather sparse (no more than 30 terms in any of the factors), and contain words such as 'year', 'fact', ''system', 'air', 'understand', 'human', 'true', 'teach' or 'fact', 'live', 'discuss', 'question' – clearly general words found in any English corpus. Interestingly, $\boldsymbol{W}$ contains also factors with terms 'christian', 'god' and 'space'. While these terms are often associated to one of the

Table 2: Average densities (fraction of non-zero elements) of the factor matrices.

Dataset	(k, p, q)	Dominated	SharedAsso	ConcatAsso	JSNMF
News	$(4, 16, 16)$	0.075	0.080	0.082	0.499
	$(8, 8, 8)$	0.080	0.090	0.096	0.611
	$(16, 4, 4)$	0.065	0.089	0.096	0.712
DBLP	$(4, 8, 20)$	0.035	0.043	0.051	0.768
	$(8, 4, 16)$	0.029	0.043	0.059	0.790
Mammals	$(4, 8, 8)$	0.247	0.234	0.224	0.860
	$(8, 4, 4)$	0.251	0.269	0.265	0.906
Web	$(4, 16, 16)$	0.030	0.030	0.024	0.850
	$(8, 8, 8)$	0.030	0.029	0.025	0.899
	$(16, 4, 4)$	0.024	0.022	0.025	0.952
Tags	$(4, 16, 16)$	0.102	0.117	0.089	0.996
	$(8, 8, 8)$	0.111	0.134	0.109	0.998
	$(16, 4, 4)$	0.115	0.122	0.109	0.999

Table 3: MDL scores for different parameter combinations. Values are scaled by 10^4.

Dataset (k, p, q)	Dominated	SharedAsso	ConcatAsso
News			
$(4, 16, 16)$	3.544	3.575	3.584
$(8, 8, 8)$	3.532	3.571	3.565
$(16, 4, 4)$	3.535	3.576	3.565
DBLP			
$(4, 8, 20)$	26.612	26.663	27.350
$(8, 4, 16)$	26.593	26.704	27.359
Mammals			
$(4, 8, 8)$	10.448	10.033	9.668
$(8, 4, 4)$	10.217	9.892	9.830
Web			
$(4, 16, 16)$	43.886	43.818	43.755
$(8, 8, 8)$	44.439	43.984	44.086
$(16, 4, 4)$	44.968	43.880	44.086
Tags			
$(4, 16, 16)$	13.423	13.588	13.858
$(8, 8, 8)$	13.354	13.611	13.800
$(16, 4, 4)$	13.714	13.622	13.800

newsgroups, they in fact are common enough to appear in both newsgroups' posts.

Matrix $\boldsymbol{U}$ contains terminology that is specific to the newsgroup `sci.space`, for example, 'planetary', 'nasa', 'rocket', 'astronomy' or 'fuel', 'chemistry', 'spacecraft', 'lunar'. Similarly, $\boldsymbol{V}$ has terms typical to discussion about Christianity: 'bibl', 'holy', 'cathol', 'scriptur', 'testam' or 'david', 'paul', 'book', 'jewish', 'condem', 'corinthian'. Notice that as the Boolean decomposition does not penalize for overlapping factors, same words appear many times: for example the term 'bibl' appears in five of eight factors in $\boldsymbol{V}$.

For Tags data, the algorithm was again `Dominated` but this time with weighting (as the matrices have different characteristics). The parameters were $k = p = q = 8$, as this was the MDL-optimal combination. The two matrices have very different density, and this can be seen from the results. The LabelMe data (in matrix $\boldsymbol{X}$) has density of about 12%, while the Flickr data (in matrix $\boldsymbol{Y}$) has much lower density (about 3%). Consequently, the specific factors in $\boldsymbol{U}$ have more elements that those in $\boldsymbol{V}$. The factors in $\boldsymbol{W}$ are in between, having generally fewer items than the factors in $\boldsymbol{U}$ but more than those in $\boldsymbol{V}$. All factors were meaningful; for example the factor having words 'sky' and 'tree' in $\boldsymbol{W}$ has tags that can be used to describe many outdoor images. Another factor had tags 'building', 'car', 'road', 'sidewalk', and 'tree' – together they fit to many images from city streets.

The factors in $\boldsymbol{U}$ were more specific than those in $\boldsymbol{W}$. An example is 'building', 'car', 'headlight', 'mirror', 'road', 'sidewalk', 'windshield', which provides more specific designation of the images.

Scalability. The algorithms were run on Linux servers with 8 hyperthreading 2.5GHz cores and 32GB of main memory. All reported times are wall-clock times.

In general, the three Boolean methods are approximately equivalently fast. With the News data, for example, they all took around 11 seconds to find the decompositions for the three different parameter combinations. `JSNMF` was considerably slower, taking over 100 seconds. An exception to this was the Mammals data, where `SharedAsso` was the fastest needing 15 seconds, `ConcatAsso` was second with 27 seconds, `JSNMF` was third with 108 seconds, and `Dominated` needed in total 1700 seconds. This increase in `Dominated`'s time is explained by the density of the Mammals data and the consequently high number of closed itemsets. Almost all of that time was spend on building the set system and solving the Max k-cover.

We also experimented increasing the minimum support for closed itemsets with Mammals data from 10% to 20%. This reduced the running time to 202 seconds with almost no effect to the error.

7 Related Work

Boolean matrix factorizations have enjoyed some amount of research interest in data mining recently. They were introduced in data mining in [10], although similar concepts, such as tiling [2], had been studied earlier. The algorithm of Belohlavek and Vychodil [1] stems from different, but closely related origin: formal concept analysis.

Joint (or shared) subspace learning is also an emerging topic in data mining and machine learning. The uses include relational learning [14], theme discovery [8], tagging [15], multi-label classification [6], and social media retrieval [4]. Here we have understood shared subspace learning in a broad sense; of the above, only [4, 8, 14] do matrix factorization. Of these, only [4] allows mixing shared and specific factors in the way we do.

8 Conclusions

We have formulated the JSBMF problem and presented three algorithms to solve it. The algorithms are, arguably, rather simple and straight forward. We consider this being a virtue of the algorithms, not a drawback: their simple construction allows for efficient implementation, as is demonstrated in the experiments. The experiments show that the algorithms work as expected, and with synthetic data they were even better than the `JSNMF` algorithm. With real-world data `JSNMF` obtained smaller reconstruction error, but also returned much denser factor matrices. The benefits of the Boolean decomposition are rarely on better reconstruction error; rather, Boolean factorizations benefit from sparse and easy-to-interpret factors. The case of joint subspace factorization seems to be no different.

This paper focuses on the methodology of doing the joint subspace Boolean factorization. While we did not study the applications, save the data exploration done using JSBMF, one could easily think that JSBMF could be applied to similar tasks as, say, JSNMF has been applied. Studying these applications is an interesting topic for the future work.

References

[1] R. Belohlavek and V. Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J Comput System Sci*, 76(1):3–20, 2010.

[2] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Discovery Science*, pages 77–122, 2004.

[3] P. D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[4] S. K. Gupta et al. Nonnegative shared subspace learning and its application to social media retrieval. In *KDD*, pages 1169–1178, 2010.

[5] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min Knowl Discov*, 8(1):53–87, 2004.

[6] S. Ji, L. Tang, S. Yu, and J. Ye. A shared-subspace learning framework for multi-label classification. *ACM Trans Knowl Discov Data*, 4(2), 2010.

[7] D. S. Johnson. Approximation algorithms for combinatorial problems. *J Comput System Sci*, 9:256–278, 1974.

[8] Y. Lin, H. Sundaram, et al. Temporal patterns in social media streams: Theme discovery and evolution using joint analysis of content and context. In *Multimedia and Expo*, 2009.

[9] P. Miettinen. Sparse boolean matrix factorizations. In *ICDM*, pages 935–940, 2010.

[10] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Trans Knowl Data Eng*, 20(10):1348–1362, 2008.

[11] P. Miettinen and J. Vreeken. Model order selection for Boolean matrix factorization. In *KDD*, pages 51–59, 2011.

[12] A. J. Mitchell-Jones et al. *The Atlas of European Mammals*. Academic Press, 1999.

[13] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: A Database and Web-Based Tool for Image Annotation. *Int J Comput Vis*, 77(1-3):157–173, 2008.

[14] A. Singh and G. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.

[15] L. Wu, L. Yang, N. Yu, and X.-S. Hua. Learning to tag. In *WWW*, pages 361–370, 2009.